

# *Validación avanzada*

## lección 6

De momento solo hemos hecho validaciones básicas usando la anotación `@Required`. A veces es necesario escribir nuestra propia lógica de validación. En el apéndice C se explica como funciona la validación. Aquí vamos a añadir validaciones con lógica propia a tu aplicación.

## 6.1 Alternativas de validación

Vamos a refinar tu código para que el usuario no pueda asignar pedidos a una factura si los pedidos no han sido servidos todavía. Es decir, solo los pedidos servidos pueden asociarse a una factura. Aprovecharemos la oportunidad para explorar diferentes formas de hacer esta validación.

### 6.1.1 Añadir la propiedad `delivered` a `Order`

Para hacer esto, lo primero es añadir una nueva propiedad a la entidad `Order`. La propiedad `delivered` (listado 6.1).

**Listado 6.1 Nueva propiedad `delivered` en la entidad `Order`**

```
private boolean delivered;

public boolean isDelivered() {
    return delivered;
}

public void setDelivered(boolean delivered) {
    this.delivered = delivered;
}
```

Ahora actualiza el esquema de la base de datos y ejecuta las sentencias SQL del listado 6.2 contra la base de datos. Puedes hacerlo con la perspectiva de base de datos de Eclipse (ver lección 1).

**Listado 6.2 Poner a false la columna `delivered` de la tabla `CommercialDocument`**

```
update CommercialDocument
set delivered = false
```

Además es necesario añadir la propiedad `delivered` a la vista. Modifica la vista `Order` como muestra el listado 6.3.

**Listado 6.3 Vista de `Order` modificada para incluir la propiedad `delivered`**

```
@Views({
    @View( extendsView="super.DEFAULT",
        members="delivered; invoice { invoice }" // delivered añadida
    ),
    ...
})
public class Order extends CommercialDocument {
```

Ahora tienes una nueva propiedad `delivered` que el usuario puede marcar para indicar que el pedido ha sido servido. Ejecuta el nuevo código y marca algunos de los pedidos existentes como servidos.

### 6.1.2 Validar con `@EntityValidator`

En tu aplicación actual el usuario puede añadir cualquier pedido que le plazca a una factura usando el módulo `Invoice`, y puede asignar una factura a cualquier pedido desde el módulo `Order`. Vamos a restringir esto. Solo los pedidos servidos podrán añadirse a una factura.

La primera alternativa que usaremos para implementar esta validación es mediante `@EntityValidator`. Esta anotación te permite asignar a tu entidad una clase con la lógica de validación deseada. Anotemos tu entidad `Order` tal como muestra el listado 6.4.

**Listado 6.4 `@EntityValidator` para la entidad `Order`**

```
@EntityValidator(
    value=DeliveredToBeInInvoiceValidator.class, // Clase con la lógica de validación
    properties= {
        @PropertyValue(name="year"), // El contenido de estas propiedades
        @PropertyValue(name="number"), // se mueve desde la entidad Order
        @PropertyValue(name="invoice"), // al validador antes de
        @PropertyValue(name="delivered") // ejecutar la validación
    }
)
public class Order extends CommercialDocument {
```

Cada vez que un objeto `Order` se crea o modifica un objeto del tipo `DeliveredToBeInInvoiceValidator` es creado, entonces las propiedades `year`, `number`, `invoice` y `delivered` se rellenan con las propiedades del mismo nombre del objeto `Order`. Después de eso, el método `validate()` del validador se ejecuta. Puedes ver el código del validador en el listado 6.5.

**Listado 6.5 Validador para que el pedido esté entregado para estar en una factura**

```
package org.openxava.invoicing.validators; // En el paquete 'validators'

import org.openxava.invoicing.model.*;
import org.openxava.util.*;
import org.openxava.validators.*;

public class DeliveredToBeInInvoiceValidator
    implements IValidator { // Ha de implementar IValidator

    private int year; // Propiedades a ser inyectada desde Order
    private int number;
    private boolean delivered;
    private Invoice invoice;

    public void validate(Messages errors) // La lógica de validación
        throws Exception
```

```

{
    if (invoice == null) return;
    if (!delivered) {
        errors.add( // Al añadir mensajes a errores la validación fallará
                    "order_must_be_delivered", // Un id del archivo i18n
                    year, number); // Argumentos para el mensaje
    }
}

// Getters y setters para year, number, delivered y invoice
...
}
    
```

La lógica de validación es extremadamente fácil, si una factura está presente y este pedido no ha sido servido añadimos un mensaje de error, por tanto la validación fallará. Has de añadir el mensaje de error en el archivo *Invoicing/i18n/Invoicing-messages\_en.properties*. Tal como muestra el listado 6.6.

#### Listado 6.6 Internacionalización del error en Invoicing-messages\_en.properties

```

# Messages for the Invoicing application
order_must_be_delivered=Order {0}/{1} must be delivered in order to be added to
an Invoice
    
```

Ahora puedes intentar añadir pedidos a una factura con la aplicación, verás como los pedidos no servidos son rechazados. Como se ve en la figura 6.1.

The screenshot shows the 'Orders' tab in the application. A table titled 'Add elements to 'Orders of Invoice'' contains the following data:

	Year	Number	Delivered	
<input type="checkbox"/>	2009	3	Yes	7/2
<input checked="" type="checkbox"/>	2009	4	Yes	8/1
<input checked="" type="checkbox"/>	2009	2	No	7/2
<input type="checkbox"/>	2009	5	No	8/1

Annotations in the figure:

- 1:** Points to the 'Add' button in the top left.
- 2:** Points to the 'Add' button in the table.
- 3:** Points to the error messages and the confirmation message at the bottom.

Error messages displayed:

- Order 2,009/2 must be delivered in order to be added to an Invoice
- ERROR! 1 element(s) NOT added to Orders of Invoice

Confirmation message:

- 1 element(s) added to Orders of Invoice

**Figura 6.1 Añadir pedidos no entregados produce errores de validación**

Ya tienes tu validación hecha con `@EntityValidator`. No es difícil, pero es un poco verboso, porque necesitas escribir una clase nueva solo para añadir 2 líneas de lógica. Aprendamos otras formas de hacer esta misma validación.

### 6.1.3 Validar con métodos de retrollamada JPA

Vamos a probar otra forma más sencilla de hacer esta validación, simplemente

moviendo la lógica de validación desde la clase validador a la misma entidad Order, en este caso a un método @PreUpdate.

Lo primero es eliminar la clase DeliveredToBeInInvoiceValidator de tu proyecto. También quita la anotación @EntityValidator de tu entidad Order (listado 6.7).

#### Listado 6.7 Quitar la anotación @EntityValidator de la entidad Order

```
@EntityValidator(value=DeliveredToBeInInvoiceValidator.class,
    properties= { // Quita la anotación @EntityValidator
        @PropertyValue(name="year"),
        @PropertyValue(name="number"),
        @PropertyValue(name="invoice"),
        @PropertyValue(name="delivered")
    }
}

+
public class Order extends CommercialDocument {
```

Acabamos de eliminar la validación. Ahora, vamos a añadirla de nuevo, pero ahora dentro de la misma clase Order. Escribe el método validate() del listado 6.8 en tu clase Order.

#### Listado 6.8 Método de retrollamada JPA para validar en la entidad Order

```
@PreUpdate // Justo antes de actualizar la base de datos
private void validate() throws Exception {
    if (invoice != null && !isDelivered()) { // La lógica de validación
        throw new InvalidStateException( // La excepción de validación del
            new InvalidValue[] { // marco de validación Hibernate Validator
                new InvalidValue(
                    "Order must be delivered",
                    getClass(), "delivered",
                    true, this)
            }
        );
    }
}
```

Antes de grabar un pedido esta validación se ejecutará, si falla una InvalidStateException será lanzada. Esta excepción es del marco de validación Hibernate Validator, de esta forma OpenXava sabe que es una excepción de validación. Lo engorroso de esta solución es que InvalidStateException requiere un *array* de objetos InvalidValue. Lo bueno es que con solo un método dentro de tu entidad tienes la validación hecha.

### 6.1.4 Validar en el setter

Otra alternativa para hacer tu validación es poner tu lógica de validación dentro del método *setter*. Es un enfoque simple y llano. Para probarlo, quita el método validate() de tu entidad Order, y modifica el método setInvoice() de la forma que ve en listado 6.9.

**Listado 6.9 Validación dentro de un método setter de invoice en Order**

```

public void setInvoice(Invoice invoice) {
    if (invoice != null && !isDelivered()) { // La lógica de validación
        throw new InvalidStateException( // La excepción de Hibernate Validator
            new InvalidValue[] {
                new InvalidValue(
                    "Order must be delivered",
                    getClass(), "delivered",
                    true, this)
            }
        );
    }
    this.invoice = invoice; // La asignación típica del setter
}

```

Esto funciona exactamente como las dos opciones anteriores. Es parecida a la alternativa del `@PrePersist`, solo que no depende de JPA, es una implementación básica de Java.

**6.1.5 Validar con Hibernate Validator**

Como opción final vamos a hacer la más breve. Consiste en poner tu lógica de validación dentro de un método booleano anotado con la anotación de Hibernate Validator `@AssertTrue`.

Para implementar esta alternativa primero quita la lógica de validación del método `setInvoice()`. Después, añade `isDeliveredToBeInInvoice()` del listado 6.10 a tu entidad `Order`.

**Listado 6.10 Validar Order usando una anotación `@AssertTrue`**

```

@AssertTrue // Antes de grabar confirma que el método devuelve true, si no lanza una excepción
private boolean isDeliveredToBeInInvoice() {
    return invoice == null || isDelivered(); // La lógica de validación
}

```

Esta es la forma más simple de validar, porque solo anotamos el método con la validación, y es Hibernate Validator el responsable de llamar este método al grabar, y lanzar la `InvalidStateException` correspondiente si la validación no pasa.

**6.1.6 Validar al borrar con `@RemoveValidator`**

Las validaciones que hemos visto hasta ahora se hacen cuando la entidad se modifica, pero a veces es útil hacer la validación justo al borrar la entidad, y usar la validación para vetar el borrado de la misma.

Vamos a modificar la aplicación para impedir que un usuario borre un pedido si éste tiene una factura asociada. Para hacer esto anota tu entidad `Order` con

## 111 Lección 6: Validación avanzada

@RemoveValidator, como muestra el listado 6.11.

### Listado 6.11 @RemoveValidator para la entidad Order

```
@RemoveValidator(OrderRemoveValidator.class) // La clase con la validación
public class Order extends CommercialDocument {
```

Ahora, antes de borrar un pedido la lógica de OrderRemoveValidator se ejecuta, y si la validación falla el pedido no se borra. Veamos el código de este validador en el listado 6.12.

### Listado 6.12 Validador para validar si un pedido puede borrarse

```
package org.openxava.invoicing.validators; // En el paquete 'validators'

import org.openxava.invoicing.model.*;
import org.openxava.util.*;
import org.openxava.validators.*;

public class OrderRemoveValidator
    implements IRemoveValidator { // Ha de implementar IRemoveValidator

    private Order order;

    public void setEntity(Object entity) // La entidad a borrar se inyectará
        throws Exception // con este método antes de la validación
    {
        this.order = (Order) entity;
    }

    public void validate(Messages errors) // La lógica de validación
        throws Exception
    {
        if (order.getInvoice() != null) {
            errors.add("cannot_delete_order_with_invoice"); // Añadiendo mensajes
                                                             // a errors la validación fallará y el borrado se abortará
        }
    }
}
```

La lógica de validación está en el método validate(). Antes de llamarlo la entidad a validar es inyectada usando setEntity(). Si se añaden mensajes al objeto errors la validación fallará y la entidad no se borrará. Has de añadir el mensaje de error en el archivo *Invoicing/i18n/Invoicing-messages\_en.properties*. Véase el listado 6.13.

### Listado 6.13 Internacionalización del error en Invoicing-messages\_en.properties

```
cannot_delete_order_with_invoice=Order with invoice cannot be deleted
```

Ahora si intentas borrar un pedido con una factura asociada obtendrás un mensaje de error y el borrado no se producirá.

Puedes ver que usar un @RemoveValidator no es difícil, pero es un poco verboso. Has de escribir una clase nueva solo para añadir un simple “if”.

Examinemos una alternativa más breve.

### 6.1.7 Validar al borrar con un método de retrollamada JPA

Vamos a probar otra forma más simple de hacer esta validación al borrar, moviendo la lógica de validación desde la clase validador a la misma entidad `Order`, en este caso en un método `@PreRemove`.

El primer paso es eliminar la clase `OrderRemoveValidator` de tu proyecto. Además quita la anotación `@RemoveValidator` de tu entidad `Order` (listado 9.14).

#### Listado 6.14 Quitar la anotación `@RemoveValidator` de de la entidad `Order`

```
@RemoveValidator(OrderRemoveValidator.class) // Quitamos @RemoveValidator
public class Order extends CommercialDocument {
```

Hemos quitado la validación. Añadámosla otra vez, pero ahora dentro de la misma clase `Order`. Añade el método `validateOnRemove()` del listado 6.15 a la clase `Order`.

#### Listado 6.15 Método de retrollamada JPA para validar al borrar en `Order`

```
@PreRemove // Justo antes de borrar la entidad
private void validateOnRemove() {
    if (invoice != null) { // La lógica de validación
        throw new IllegalStateException( // Lanza una excepción runtime
            XavaResources.getString( // Para obtener un mensaje de texto
                "cannot_delete_order_with_invoice"));
    }
}
```

Antes de borrar un pedido esta validación se efectuará, si falla se lanzará una `IllegalStateException`. Puedes lanzar cualquier excepción *runtime* para abortar el borrado. Tan solo con un método dentro de la entidad tienes la validación hecha.

### 6.1.8 ¿Cuál es la mejor forma de validar?

Has aprendido varias formas de hacer la validación sobre tus clases del modelo. ¿Cuál de ellas es la mejor? Todas ellas son opciones válidas. Depende de tus circunstancias y preferencias personales. Si tienes una validación que no es trivial y es reutilizable en varios puntos de tu aplicación, entonces usar un `@EntityValidator` y `@RemoveValidator` es una buena opción. Por otra parte, si quieres usar tu modelo fuera de OpenXava y sin JPA, entonces el uso de la validación en los *setters* es mejor.

En nuestro caso particular hemos optado por `@AssertTrue` para la validación “el pedido ha de estar servido para estar en una factura” y por `@PreRemove` para



la validación al borrar. Ya que son las alternativas más simples que funcionan.

## 6.2 Crear tu propia anotación de Hibernate Validator

Las técnicas mencionadas hasta ahora son muy útiles para la mayoría de las validaciones de tus aplicaciones. Sin embargo, a veces te encuentras con algunas validaciones que son muy genéricas y quieres usarlas una y otra vez. En este caso definir tu propia anotación de Hibernate Validator puede ser una buena opción. Definir un Hibernate Validator es más largo y engorroso que lo que hemos visto hasta ahora, pero usarlo y reusarlo es simple, tan solo añadir una anotación a tu propiedad o clase.

Vamos a aprender como crear un Hibernate Validator.

### 6.2.1 Usar un Hibernate Validator en tu entidad

Usar un Hibernate Validator es superfácil. Simplemente anota tu propiedad, como ves en el listado 6.16.

#### Listado 6.16 Usar una anotación de Hibernate Validator para nuestra propiedad

```
@ISBN // Esta anotación indica que esta propiedad tiene que validarse como un ISBN
private String isbn;
```

Solo con añadir @ISBN a tu propiedad, y ésta será validada justo antes de que la entidad se grabe en la base de datos, ¡genial! El problema es que @ISBN no está incluida como un validador predefinido en el marco de validación Hibernate Validator. Esto no es un gran problema, si quieres una anotación @ISBN, hazla tú mismo. De hecho, vamos a crear la anotación de validación @ISBN en esta sección.

Antes de nada, añadamos una nueva propiedad isbn a Product. Edita tu clase Product y añádele el código del listado 6.17.

#### Listado 6.17 Nueva propiedad isbn en Product

```
@Column(length=10)
private String isbn;

public String getIsbn() {
    return isbn;
}

public void setIsbn(String isbn) {
    this.isbn = isbn;
}
```

Actualiza el esquema de tu base de datos, y ejecuta el módulo Product con tu navegador. Sí, la propiedad isbn ya está ahí. Ahora, puedes añadir la validación.

### 6.2.2 Definir tu propia anotación ISBN

Creemos la anotación @ISBN. Primero, crea un paquete en tu proyecto llamado `org.openxava.invoicing.annotations`, entonces sigue las instrucciones de la figura 6.2 para crear una nueva anotación llamada ISBN.

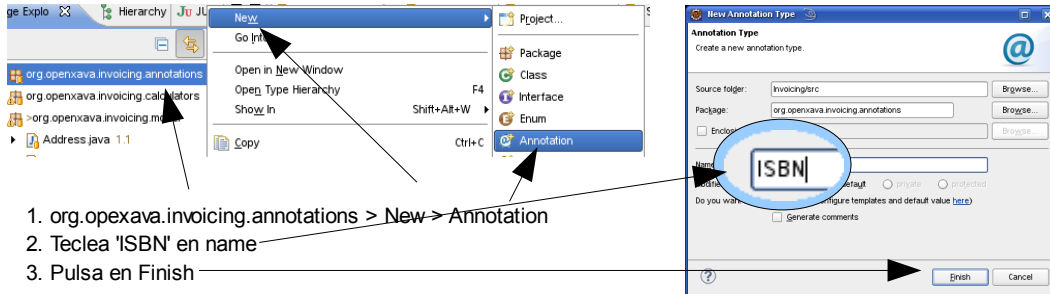


Figura 6.2 Crear la nueva anotación ISBN con Eclipse

Edita el código de tu recién creada anotación ISBN y déjala como la del listado 6.18.

#### Listado 6.18 Código para la anotación ISBN

```
package org.openxava.invoicing.annotations; // En el paquete annotations

import java.lang.annotation.*;
import org.hibernate.validator.*;
import org.openxava.invoicing.validators.*;

@ValidatorClass(ISBNValidator.class) // Esta clase contiene la lógica de validación
@Target({ ElementType.FIELD, ElementType.METHOD })
@Retention(RetentionPolicy.RUNTIME)
public @interface ISBN { // Una definición de anotación Java convencional

    String message() default "ISBN does not exist"; // El mensaje si la validación falla

}
```

Como puedes ver, es una definición de anotación normal y corriente, solo que usas `@ValidatorClass` para indicar la clase con la lógica de validación. Escribamos la clase `ISBNValidator`.

### 6.2.3 Usar Apache Commons Validator para implementar la lógica

Vamos a escribir la clase `ISBNValidator` con la lógica de validación para un ISBN. En lugar de escribir nosotros mismos la lógica para validar un ISBN usaremos el proyecto Commons Validator<sup>8</sup> de Apache. Commons Validator contiene algoritmos de validación para direcciones de correo electrónico, fechas, URL y así por el estilo. El `commons-validator.jar` se incluye por defecto en los

8 <http://commons.apache.org/validator/>

proyectos OpenXava, por tanto lo puedes usar sin ninguna configuración adicional.

El código para ISBNValidator está en el listado 6.19.

#### Listado 6.19 Versión inicial de ISBNValidator

```
package org.openxava.invoicing.validators; // En el paquete validators

import org.hibernate.validator.*;
import org.openxava.util.*;
import org.openxava.invoicing.annotations.*;

public class ISBNValidator
    implements Validator<ISBN> { // Tiene que implementar Validator<ISBN>

    private static org.apache.commons.validator.ISBNValidator
        validator = // De Commons Validator
            new org.apache.commons.validator.ISBNValidator();

    public void initialize(ISBN isbn) {
    }

    public boolean isValid(Object value) { // Contiene la lógica de validación
        if (Is.empty(value)) return true;
        return validator
            .isValid(value.toString()); // Usa Commons Validator
    }
}
```

Como ves, la clase validador tiene que implementar Validator del paquete org.hibernate.validator. Esto fuerza a tu validador a implementar initialize() e isValid(). El método isValid() contiene la lógica de validación. Fíjate que si el elemento a validar está vacío asumimos que es válido, porque validar si un valor está presente es responsabilidad de otras anotaciones, como @Required, y no de @ISBN.

En este caso la lógica de validación es sencillísima, porque nos limitamos a llamar al validador ISBN de Apache Commons Validator.

@ISBN está listo para usar. Para hacerlo anota tu propiedad isbn con él. Puedes ver cómo en el listado 6.20.

#### Listado 6.20 La propiedad isbn anotada con @ISBN

```
@Column(length=10) @ISBN
private String isbn;
```

Ahora, puedes probar tu módulo, y verificar que el ISBN que introduces se valida correctamente. Enhorabuena, has escrito tu primer Hibernate Validator. No ha sido tan difícil: una anotación, una clase.

Este @ISBN es suficientemente bueno para usarlo en la vida real, sin embargo,

vamos a mejorarlo un poco más, y así tendremos la posibilidad de experimentar con algunas posibilidades interesantes.

#### 6.2.4 Llamar a un servicio web REST para validar el ISBN

Aunque la mayoría de los validadores tienen una lógica simple, puedes crear validadores con una lógica compleja si lo necesitas. Por ejemplo, en el caso de nuestro ISBN, queremos, no solo verificar el formato correcto, sino también comprobar que existe de verdad un libro con ese ISBN. Una forma de hacer esto es usando servicios web.

Como seguramente ya sepas, un servicio web es una funcionalidad que reside en un servidor web y que tú puedes llamar desde tu programa. La forma tradicional de desarrollar servicios web es mediante los estándares WS-\*, como SOAP, UDDI, etc. Aunque, hoy en día está surgiendo una forma más simple de desarrollar servicios, REST. REST consiste básicamente en usar la ya existente “forma de trabajar” de internet para comunicación entre programas. Llamar a un servicio REST consiste en usar una URL web convencional para obtener un recurso de un servidor web. Este recurso usualmente contiene datos en formato XML, HTML, JSON, etc. En otras palabras, los programas usan internet de la misma manera que lo hacen los usuarios con sus navegadores.

Hay bastantes sitio con servicios web SOAP y REST para consultar el ISBN de un libro, pero no suele ser gratis. Por tanto, vamos a usar una alternativa más barata, que va a ser llamar a un sitio web convencional para hacer la búsqueda del ISBN, y examinar después la página resultado para determinar si la búsqueda ha funcionado. Algo así como un servicio web pseudo-REST.

Para llamar a la página web usaremos el marco de trabajo HtmlUnit<sup>9</sup>. Aunque el principal cometido de este marco de trabajo sea crear pruebas para tus aplicaciones web, puedes usarlo para leer cualquier página web. Lo usaremos porque es más fácil que otras librerías con este propósito, como por ejemplo Apache Commons HttpClient. Observa lo simple que es leer una página web con HtmlUnit en el listado 6.21.

##### Listado 6.21 Leer una página web usando HtmlUnit

```
WebClient client = new WebClient();
HtmlPage page = (HtmlPage) client.getPage("http://www.openxava.org/");
```

Después de esto, puedes usar el objeto page para manipular la página leída.

OpenXava usa HtmlUnit como marco subyacente para las pruebas, por tanto ya está incluido en OpenXava, pero no se incluye por defecto en las aplicaciones OpenXava, así que tienes que incluirlo tú mismo en tu aplicación. Para hacerlo,

<sup>9</sup> <http://htmlunit.sourceforge.net/>

copia los archivos *htmlunit.jar*, *commons-httpclient.jar*, *commons-codec.jar*, *htmlunit-core-js.jar*, *commons-lang.jar*, *xercesImpl.jar*, *xalan.jar*, *cssparser.jar*, *sac.jar* y *nekohtml.jar*, desde la carpeta *OpenXava/lib* a la carpeta *Invoicing/web/WEB-INF/lib*. Después de copiar estos archivos refresca el proyecto Invoicing pulsando F5.

Modifiquemos *ISBNValidator* para que haga uso de este servicio REST. Puedes ver el resultado en el listado 6.22.

#### Listado 6.22 ISBNValidator que usa un servicio web REST

```
package org.openxava.invoicing.validators;

import org.apache.commons.logging.*;
import org.hibernate.validator.*;
import org.openxava.util.*;
import org.openxava.invoicing.annotations.*;

import com.gargoylesoftware.htmlunit.*; // Para usar HtmlUnit
import com.gargoylesoftware.htmlunit.html.*; // Para usar HtmlUnit

public class ISBNValidator implements Validator<ISBN> {

    private static Log log = LoggerFactory.getLog(ISBNValidator.class);
    private static org.apache.commons.validator.ISBNValidator
        validator =
            new org.apache.commons.validator.ISBNValidator();

    public void initialize(ISBN isbn) {
    }

    public boolean isValid(Object value) {
        if (Is.empty(value)) return true;
        if (!validator.isValid(value.toString())) return false;
        return isbnExists(value); // Aquí hacemos la llamada REST
    }

    private boolean isbnExists(Object isbn) {
        try {
            WebClient client = new WebClient();
            HtmlPage page = (HtmlPage) client.getPage( // Llamamos a
                "http://www.bookfinder4u.com/" + // bookdiner4u
                "IsbnSearch.aspx?isbn=" + // con una URL para buscar
                isbn + "&mode=direct"); // por ISBN
            return page.asText() // Comprueba si la página resultante contiene
                .indexOf("ISBN: " + isbn) >= 0; // el ISBN buscado
        }
        catch (Exception ex) {
            log.warn("Impossible to connect to bookfinder4u" +
                "to validate the ISBN. Validation fails", ex);
            return false; // Si hay algún error asumimos que la validación ha fallado
        }
    }
}
```

Simplemente buscamos una página usando como argumento en la URL el

ISBN, si la página resultante contiene el ISBN buscado quiere decir que la búsqueda ha sido satisfactoria, si no es que la búsqueda ha fallado. El método `page.asText()` devuelve el contenido de la página HTML sin las marcas HTML, es decir, con solo la información textual.

Puedes usar este truco con cualquier sitio que te permita hacer búsquedas, así puedes consultar virtualmente millones de sitios web desde tu aplicación. En un servicio REST más puro el resultado hubiera sido un documento XML en vez de uno HTML, pero hubieras tenido que pasar por caja.

Prueba ahora tu aplicación y verás como si introduces un ISBN no existente la validación falla.

### 6.2.5 Añadir atributos a tu anotación

Creas una anotación Hibernate Validator cuando quieres reutilizar la validación varias veces, usualmente en varios proyectos. En este caso, necesitas hacer tu validación adaptable, para que sea reutilizable de verdad. Por ejemplo, en el proyecto actual buscar en [www.bookfinder4u.com](http://www.bookfinder4u.com) el ISBN es conveniente, pero en otro proyecto, o incluso en otra entidad de tu actual proyecto, puede que no quieras hacer esa búsqueda. Necesitas hacer tu anotación más flexible.

La forma de añadir esta flexibilidad a tu anotación de validación es mediante los atributos. Por ejemplo, podemos añadir un atributo de búsqueda booleano a nuestra anotación ISBN para poder escoger si queremos buscar el ISBN en internet para validar o no. Para hacerlo, simplemente añade el atributo `search` al código de la anotación ISBN, tal como muestra el listado 6.23.

#### Listado 6.23 Anotación ISBN con el atributo search

```
public @interface ISBN {

    boolean search() default true; // Para (des)activar la búsqueda web al validar
    String message() default "ISBN does not exist";

}
```

Este nuevo atributo `search` puede leerse de la clase validador. Míralo en el listado 6.24.

#### Listado 6.24 ISBNValidator con el atributo search

```
public class ISBNValidator implements Validator<ISBN> {

    ...

    private boolean search; // Almacena la opción search

}
```

```

public void initialize(ISBN isbn) { // Lee los atributos de la anotación
    this.search = isbn.search();
}

public boolean isValid(Object value) {
    if (Is.empty(value)) return true;
    if (!validator.isValid(value.toString())) return false;
    return search?isbnExists(value):true; ← Usa search
}

...
}

```

Aquí ves la utilidad del método `initialize()`, que lee la anotación para inicializar el validador. En este caso simplemente almacenamos el valor de `isbn.search()` para preguntar por él en `isValid()`.

Ahora puedes escoger si quieres llamar a nuestro servicio pseudo-REST o no para hacer la validación ISBN. Véase el listado 6.25.

#### Listado 6.25 Usar @ISBN con el atributo search

```

@ISBN(search=false) // En este caso no se hace un búsqueda en la web para validar el ISBN
private String isbn;

```

Usando esta técnica puedes añadir cualquier atributo que necesites para dar más flexibilidad a tu anotación ISBN.

¡Enhorabuena! Has aprendido como crear tu propia anotación de Hibernate Validator, y de paso a usar la útil herramienta HtmlUnit.

## 6.3 Pruebas JUnit

Nuestra meta no es desarrollar una ingente cantidad de código, sino crear software de calidad. Al final, si creas software de calidad acabarás escribiendo más cantidad de software, porque podrás dedicar más tiempo a hacer cosas nuevas y excitantes, y menos depurando legiones de *bugs*. Y tú sabes que la única forma de conseguir calidad es mediante las pruebas automáticas, por tanto actualicemos nuestro código de prueba.

### 6.3.1 Probar la validación al añadir a una colección

Recuerda que hemos refinado tu código para que el usuario no pueda asignar pedidos a una factura si los pedidos no están servidos. Después de esto, tu actual `testAddOrders()` de `InvoiceTest` puede fallar, porque trata de añadir el primer pedido, y es posible que ese primer pedido no esté marcado como servido.

Modifiquemos la prueba para que funcione y también para comprobar la nueva funcionalidad de validación. Mira el listado 6.26.

**Listado 6.26 testAddOrders() de InvoiceTest prueba la validación en los pedidos**

```
public void testAddOrders() throws Exception {
    assertListNotEmpty();
    execute("List.orderBy", "property=number");
    execute("Mode.detailAndFirst");
    execute("Sections.change", "activeSection=1");
    assertCollectionRowCount("orders", 0);
    execute("Collection.add",
        "viewObject=xava_view_section1_orders");
    execute("AddToCollection.add", "row=0"); // Ahora no seleccionamos al azar

    checkFirstOrderWithDeliveredEquals("Yes"); // Selecciona un pedido entregado
    checkFirstOrderWithDeliveredEquals("No"); // Selecciona uno no entregado
    execute("AddToCollection.add"); // Tratamos de añadir ambos
    assertError( // Un error, porque el pedido no entregado no se puede añadir
        "ERROR! 1 element(s) NOT added to Orders of Invoice");
    assertMessage(// Un mensaje de confirmación, porque el pedido entregado ha sido añadido
        "1 element(s) added to Orders of Invoice");

    assertCollectionRowCount("orders", 1);
    checkRowCollection("orders", 0);
    execute("Collection.removeSelected",
        "viewObject=xava_view_section1_orders");
    assertCollectionRowCount("orders", 0);
}
```

Hemos modificado la parte de la selección de pedidos a añadir, antes seleccionábamos el primero, no importaba si estaba servido o no. Ahora seleccionamos un pedido servido y otro no servido, de esta forma comprobamos que el pedido servido se añade y el no servido es rechazado.

La pieza que nos falta es la forma de seleccionar los pedidos. Esto es el trabajo del método `checkFirstOrderWithDeliveredEquals()`. Veámoslo en el listado 6.27.

**Listado 6.27 Método para marcar los pedidos seleccionándolos por 'delivered'**

```
private void checkFirstOrderWithDeliveredEquals(String value)
    throws Exception
{
    int c = getListRowCount(); // El total de filas visualizadas en la lista
    for (int i=0; i<c; i++) {
        if (value.equals(
            getValueInList(i, 2))) // 2 es la columna 'delivered'
        {
            checkRow(i);
            return;
        }
    }
    fail("Must be at least one row with delivered=" + value);
}
```

Aquí ves una buena técnica para hacer un bucle sobre los elementos



visualizados de una lista para seleccionarlos y coger algunos datos, o cualquier otra cosa que quieras hacer con los datos de la lista.

### 6.3.2 Probar validación al asignar una referencia y al borrar

Desde el módulo Invoice el usuario no pueda asignar pedidos a una factura si los pedidos no están servidos, por lo tanto, desde el módulo Order el usuario tampoco debe poder asignar una factura a un pedido si éste no está servido. Es decir, hemos de probar también la otra parte de la asociación. Lo haremos modificando el actual `testSetInvoice()` de `OrderTest`.

Además, aprovecharemos este caso para probar la validación al borrar que vimos en las secciones 9.1.6 y 9.1.7. Allí modificamos la aplicación para impedir que un usuario borrara un pedido si éste tenía una factura asociada. Ahora probaremos este hecho.

Todo esto está en el `testSetInvoice()` mejorado que puedes ver en el listado 9.28.

#### Listado 6.28 `testSetInvoice()` prueba las validaciones al grabar y al borrar

```
public void testSetInvoice() throws Exception {
    assertListNotEmpty();
    execute("List.orderBy", "property=number"); // Establece el orden de la lista
    execute("Mode.detailAndFirst");
    assertValue("delivered", "false"); // El pedido no ha de estar entregado
    execute("Sections.change", "activeSection=1");
    assertValue("invoice.number", "");
    assertValue("invoice.year", "");
    execute("Reference.search",
        "keyProperty=invoice.year");
    String year = getValueInList(0, "year");
    String number = getValueInList(0, "number");
    execute("ReferenceSearch.choose", "row=0");
    assertValue("invoice.year", year);
    assertValue("invoice.number", number);

    // Los pedidos no entregados no pueden tener factura
    execute("CRUD.save");
    assertErrorsCount(1); // No podemos grabar porque no ha sido entregado
    setValue("delivered", "true");
    execute("CRUD.save"); // Con delivered=true podemos grabar el pedido
    assertNoErrors();

    // Un pedido con factura no se puede borrar
    execute("Mode.list"); // Vamos al modo lista y
    execute("Mode.detailAndFirst"); // volvemos a detalle para cargar el pedido grabado
    execute("CRUD.delete"); // No podemos borrar porque tiene una factura asociada
    assertErrorsCount(1);

    // Restaurar los valores originales
    setValue("delivered", "false");
    setValue("invoice.year", "");
    execute("CRUD.save");
}
```

```

    assertNoErrors();
}

```

La prueba original solo buscaba una factura, ni siquiera intentaba grabarla. Ahora, hemos añadido código al final para probar la grabación de un pedido marcado como servido, y marcado como no servido, de esta forma comprobamos la validación. Después de eso, tratamos de borrar el pedido, el cual tiene una factura, así probamos también la validación al borrar.

### 6.3.3 Probar el Hibernate Validator propio

Solo nos queda probar tu Hibernate Validator ISBN, el cual usa un servicio REST para hacer la validación. Simplemente hemos de escribir una prueba que trate de asignar un ISBN incorrecto, uno inexistente y uno correcto a un producto, y ver que pasa. Para hacer esto añadamos un método `testISBNValidator()` a `ProductTest`. Lo puedes ver en el listado 6.29.

**Listado 6.29 testISBNValidator() de ProductTest prueba el Hibernate Validator**

```

public void testISBNValidator() throws Exception {
    // Buscar product1
    execute("CRUD.new");
    setValue("number", Integer.toString(product1.getNumber()));
    execute("CRUD.refresh");
    assertValue("description", "JUNIT Product 1");
    assertValue("isbn", "");

    // Con un formato de ISBN incorrecto
    setValue("isbn", "1111");
    execute("CRUD.save"); // Falla por el formato (apache commons validator)
    assertError("1111 is not a valid value for Isbn of " +
        "Product: ISBN does not exist");

    // ISBN no existe aunque tiene un formato correcto
    setValue("isbn", "1234367890");
    execute("CRUD.save"); // Falla porque no existe (el servicio REST)
    assertError("1234367890 is not a valid value for Isbn of " +
        "Product: ISBN does not exist");

    // ISBN existe
    setValue("isbn", "0932633439");
    execute("CRUD.save"); // No falla
    assertNoErrors();
}

```

Seguramente la prueba manual que hacías mientras estabas escribiendo el validador `@ISBN` era parecida a esta. Por eso, si hubieras escrito tu código de prueba antes que el código de la aplicación<sup>10</sup>, lo hubieras podido usar mientras que desarrollabas, lo cual es más eficiente que repetir una y otra vez a mano las

<sup>10</sup> Ventajas de hacer primero las pruebas:

<http://www.extremeprogramming.org/rules/testfirst.html>

pruebas con el navegador.

Fíjate que si usas `@ISBN(search=false)` esta prueba no funciona porque no solo comprueba el formato sino que también hace la búsqueda con el servicio REST. Por tanto, has de usar `@ISBN` sin atributos para anotar la propiedad `isbn` y poder ejecutar esta prueba.

Ahora ejecuta todas las prueba de tu aplicación Invoicing para verificar que todo sigue en su sitio.

## **6.4 Resumen**

En esta lección has aprendido varias formas de hacer validación en una aplicación OpenXava. Además, ahora estás preparado para encapsular toda la lógica de validación reutilizable en anotaciones usando Hibernate Validator.

La validación es una parte importante de la lógica de tu aplicación, y te ánimo a que la pongas en el modelo, es decir en las entidades; tal y como esta lección ha mostrado. Aun así, a veces es conveniente poner algo de lógica fuera de las clases del modelo. Aprenderás a hacer esto en las siguientes lecciones.